Integración del control activo en la simulación dinámica de sistemas multicuerpo

M. Á. Naya, J. Cuadrado

Escuela Politécnica Superior, Universidad de A Coruña
Mendizábal s/n, 15403 Ferrol
Tel: 981337400 ext. 3873. E-mail: minaya@cdf.udc.es

Resumen
En este artículo se presenta la introducción del control en la simulación de un vehículo automóvil. Para el desarrollo del control se ha empleado *fuzzy logic*. Con este fin se ha utilizado la *toolbox* de Matlab correspondiente. Se discuten las diferentes posibilidades de comunicar el software comercial de Matlab con el lenguaje Fortran en el que está desarrollada la simulación. Finalmente, se presentan dos maniobras de control efectuadas con esta técnica.

Palabras Clave: sistemas multicuerpo, simulación, control, lógica difusa.

Abstract
In this paper, control is introduced in an automotive vehicle simulation. Fuzzy logic is employed for the development of the control strategy. With this objective, the fuzzy logic toolbox of commercial software Matlab has been used. The different options to communicate Matlab software and Fortran programming language are discussed. Finally, two control manoeuvres implemented with these techniques are exposed.

Keywords: multibody systems, simulation, control, fuzzy logic.

1. Introducción

Durante los últimos años nuestro grupo ha examinado y mejorado diferentes métodos propuestos en la literatura para la dinámica de sistemas multicuerpo [1, 2]. Como fruto de este trabajo, se ha desarrollado un método robusto y eficiente [3]: una formulación Lagrangiana aumentada (index-3) con proyecciones de velocidad y aceleración, que incorpora la tecnología de matrices sparse. Como integrador se utiliza un método implícito de paso simple: la regla trapezoidal. El método ha mostrado un comportamiento excelente ante configuraciones singulares, topologías cambiantes o sistemas con gran rigidez numérica. Utilizando para el cálculo PCs convencionales, se alcanza el tiempo real en simulaciones de un modelo realista de vehículo automóvil ante distintas maniobras exigentes, tales como un slalom, desniveles pronunciados y escalones. Además, el método puede considerar tanto sólidos rígidos como flexibles.

Tras esto, nos hemos propuesto verificar la fiabilidad del método en el desarrollo de estrategias de control para sistemas multicuerpo reales. Estas estrategias de control se desarrollan y ponen a punto en el modelo virtual del sistema mecánico correspondiente. En concreto, nos hemos centrado en el desarrollo de algoritmos para la ejecución de dos maniobras automáticas del automóvil, empleando lógica difusa (*fuzzy logic*) en el control.

2. Fuzzy Logic

Fuzzy logic, en sentido amplio, se basa en el establecimiento de reglas de pertenencia difusas a un conjunto [4]. Las funciones de pertenencia pueden tomar valores entre 0 y 1, correspondientes a grados de pertenencia que van desde "no pertenencia" hasta "pertenencia completa". De esta manera, se refleja bastante bien la manera de expresarnos en el lenguaje común. Por ejemplo, al hablar de la estatura de una persona decimos que es alta, muy alta, un poco baja, etc. En los conjuntos de la teoría clásica tendríamos que establecer una barrera, por ejemplo 1,7 m, y entonces las personas que midan más de 1,7 m serían altas, y las que midan menos, bajas. Sin embargo, no se distinguiría entre una persona baja que mida 1,69 m y otra que mida 1,45 m. En lógica difusa, se puede decir que una persona tiene un grado de pertenencia de 0,7 al grupo de la gente alta, etc.

Por otro lado, al actuar sobre un sistema, muchas veces es necesario que dicha actuación no sea de todo o nada, o pertenezca, en sentido estricto, a unos conjuntos. En el caso de la conducción, por ejemplo, a veces será necesario "frenar un poco" o "acelerar mucho".

Las reglas de actuación en *fuzzy logic* son fáciles de expresar, dependiendo de las variables y de las actuaciones posibles. En el caso del automóvil, si nuestras variables son la velocidad y la aceleración y, actuando sobre el acelerador y el freno, queremos que el coche viaje a una velocidad media constante, las reglas de actuación tienen el aspecto siguiente: *si la velocidad es alta y la aceleración es alta, frena mucho y acelera nada*, o, *si la velocidad es media y la aceleración es nula, frena nada y acelera nada*. Con las reglas de pertenencia y las reglas de actuación, en el fondo, se genera una hiper-superficie que relaciona el valor de las variables con el de los actuadores. Esto hace que el empleo de *fuzzy logic* para el control de las maniobras de un vehículo parezca

acertado. Por otro lado, Matlab dispone de una *toolbox* de *fuzzy logic*, que facilita enormemente el uso de esta lógica de control.

3. Introducción del control en el modelo

La introducción del control en el modelo supone comunicar Matlab y el lenguaje de programación en el que está implementada la simulación. Dada su mayor rapidez, el código de simulación está escrito en Fortran. El problema fundamental es que Matlab trabaja siempre en *doble precisión*, guardando cada dato en 8 bytes, con 15 cifras decimales exactas, por lo que habrá que transformar los distintos tipos utilizados en Fortran (integer, real, ...) al propio de Matlab. Hemos estudiado tres posibilidades de comunicación entre Matlab y Fortran. En las dos primeras, es necesario el uso de un compilador de Fortran que sea compatible con Matlab. Para la tercera, es necesario un compilador de C/C++ compatible con Matlab.

a) Matlab Engine.

b) Ficheros MEX.

c) Matlab Compiler.

3.1. Empleo de Matlab Engine

La comunicación entre Matlab y Fortran a través de Matlab Engine se basa en abrir un canal de comunicación desde el lenguaje Fortran a Matlab. Matlab dispone de varias rutinas para la compilación que hacen posible la transferencia de datos entre Fortran y Matlab, así como ejecutar funciones en Matlab [5].

De lo expuesto se deriva que, aunque el programa principal de la simulación se ejecute desde la ventana de comandos del sistema operativo, iniciará una sesión en Matlab. Esto conlleva un tiempo de retardo. De igual manera, la transferencia de datos por un canal de comunicación y la ejecución de sentencias en Matlab ralentizan sensiblemente el programa. En efecto, para ejecutar un programa o función en Matlab, en el fondo, lo que se hace es escribir y ejecutar en la ventana de comandos de Matlab, a través del canal de comunicación. Sin embargo, los tiempos obtenidos de simulación no son demasiado

elevados (aproximadamente dos veces el tiempo real empleando un PC con procesador Pentium IV a 1,7 GHz y sistema operativo Windows 2000), por lo que constituye una herramienta adecuada para comprobar los algoritmos de control desarrollados.

3.2. Ficheros MEX

Matlab dispone de capacidad para escribir nuevas funciones mediante los llamados *ficheros MEX*. La aplicación correcta de esta posibilidad permite escribir en Matlab todo el programa salvo los cuellos de botella. Estos se pueden programar en Fortran o C, y ejecutarlos directamente desde Matlab como si fuese una función propia de Matlab. Se dispone de una librería de comandos que permiten comunicar Matlab con el lenguaje de programación y se compila desde Matlab.

Como se ve, la actuación es la contraria a la que se sigue utilizando Matlab Engine. Sin embargo, en este estudio se ha procedido a convertir todo el programa en un fichero MEX ejecutable en Matlab. Esto permite acceder a los ficheros de control de la *toolbox* de *fuzzy logic* aunque no resulte la solución óptima para este problema concreto.

Sin embargo, el tiempo de ejecución apenas es ligeramente superior al obtenido con Matlab Engine, y continúa siendo del orden de dos veces el tiempo real.

3.3. Matlab Compiler

Esta última técnica es sustancialmente distinta a las anteriores. Matlab Compiler es una herramienta que permite obtener lenguaje C o C++ a partir de rutinas escritas en Matlab. Por tanto, se origina aquí una nueva dificultad, que es la de mezclar lo programado en Fortran con el lenguaje C/C++ procedente de Matlab. En este momento, estamos trabajando para salvar esta dificultad, pero pensamos que es factible construyendo una librería en C/C++ a partir de Matlab que se ensambla en la fase de *linkado* con el código Fortran.

4. Maniobras desarrolladas en la simulación

Hemos desarrollado dos maniobras en las que interviene el control. En la primera se trata de que el vehículo, partiendo del reposo, avance 20 metros en línea recta y quede parado tras recorrer esa distancia. El control se encarga de la actuación sobre los pedales de acelerador y freno, así como sobre la palanca de cambios. Los valores que devuelve el controlador para los pedales varían entre 0 y 1, que corresponderían a los recorridos nulo y máximo del pedal respectivamente. La actuación sobre la palanca de cambio devuelve 0 ó 1, el primero corresponde al punto muerto y el segundo a la de avance o *drive* (se trata de un motor de cambio automático).

En la Figura 1a se presenta la actuación de los pedales, y en la Figura 1b el comportamiento del vehículo.

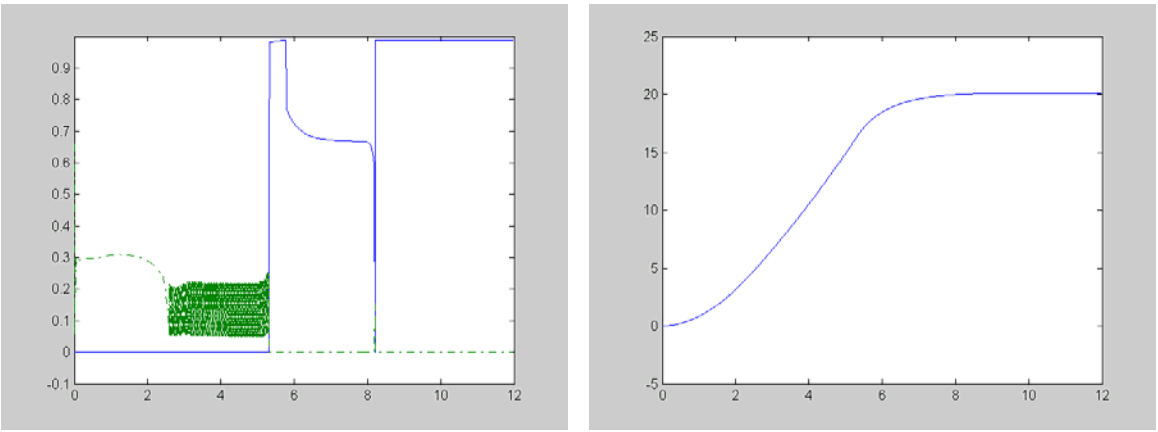

Figura 1. *a) Comportamiento de los pedales de freno (-) y acelerador (-.).*
b) Trayectoria seguida por el vehículo.

La segunda maniobra consiste en que el vehículo siga una trayectoria determinada, consistente en un tramo recto, un tramo senoidal y otro tramo recto final. En este caso el control se ha realizado sobre el volante por un lado, y sobre el acelerador y el freno por otro. El objetivo es que, al igual que un conductor real, el vehículo disminuya la velocidad cuando el error en el seguimiento de la trayectoria aumente.

En la Figura 2, que representa la trayectoria seguida por el vehículo, puede observarse que los errores obtenidos en el seguimiento de la posición son inferiores a 0,5 m. Por otro lado, si se comparan las Figuras 2b y 3b se observa cómo efectivamente la velocidad disminuye al aumentar el error.

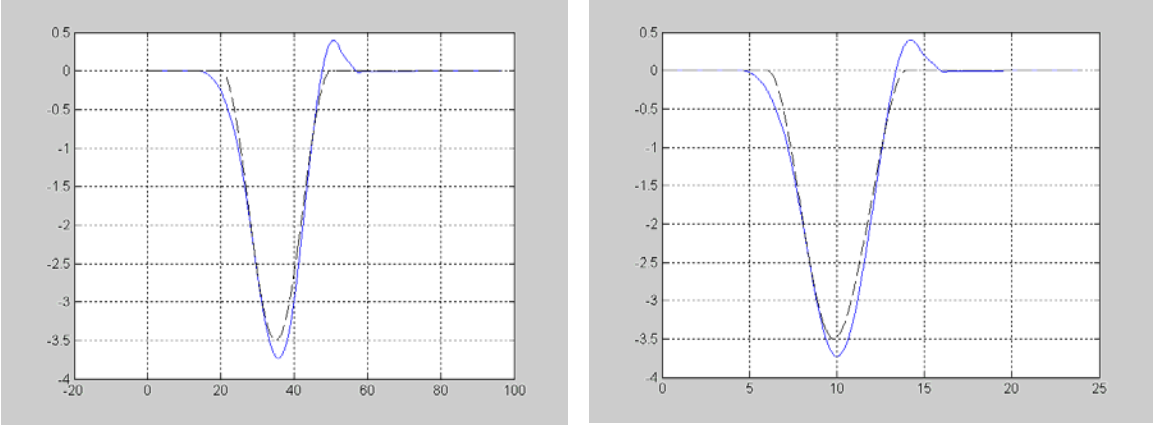

***Figura 2**. Trayectoria del vehículo:(-) efectuada (- -) deseada . a) Posición en y frente a x. b) Posición en y frente al tiempo.*

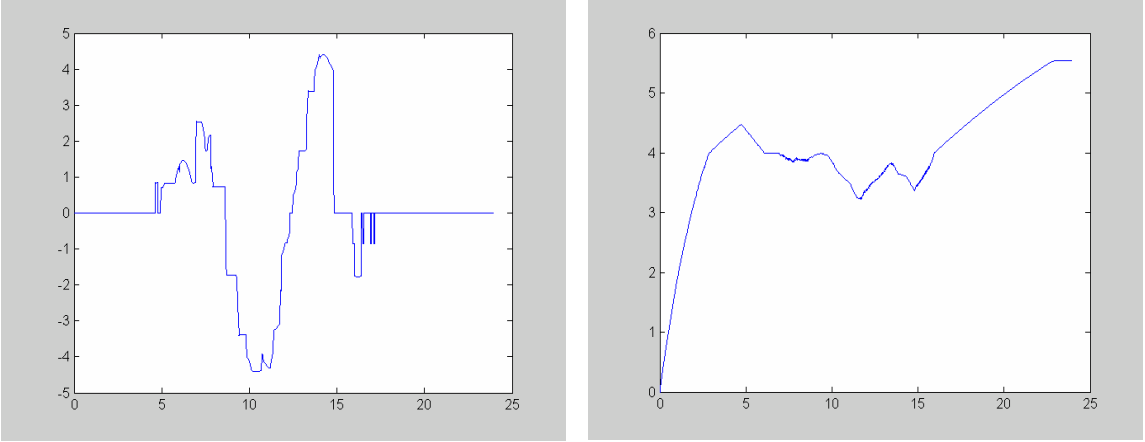

***Figura 3**. a) Giro del volante (rad) frente al tiempo. b) Evolución de la velocidad (m/s) frente al tiempo*

5. Conclusiones

Ambas maniobras se han implementado con las dos primeras técnicas de las tres anteriormente enunciadas. A la vista de los resultados, cabría señalar las siguientes conclusiones:

- Los tiempos empleados en la resolución del problema son aceptables en las dos alternativas: dos veces el valor del tiempo real.

- El empleo de Matlab Engine parece el más adecuado, puesto que el tiempo de duración de la simulación es ligeramente inferior pero, sobre todo, implica añadir menos código al programa. En esta alternativa, únicamente se deben añadir los comandos referidos a la apertura y cierre del canal de comunicación, así como los relativos a la transferencia de datos. Conceptualmente, es una solución adecuada a la necesidad de que sólo algunas partes del programa se ejecuten en Matlab.

- Convertir todo en un fichero MEX, además del código a añadir en Matlab Engine, supone introducir un nuevo fichero de cabecera que posibilita la llamada al programa desde Matlab. También se ha indicado anteriormente que esta posibilidad está orientada fundamentalmente a aquellos casos en los que todo el programa se escribe en Matlab salvo algunas partes cuya ejecución sea más rápida si se escribe C o Fortran.

6. Referencias

1. J. Cuadrado et al., "Modeling and Solution Methods for Efficient Real-Time Simulation of Multibody Dynamics", *Multibody System Dynamics*, **Vol.** (1) (1997) p. 259.

2. J. Cuadrado et al., "Intelligent Simulation of Multibody Dynamics: Space-State and Descriptor Methods in Sequential and Parallel Computing Environments", *Multibody System Dynamics*, **Vol.** (4) (2000) p. 55.

3. J. Cuadrado et al., "A Comparison in Terms of Accuracy and Efficiency between a MBS Dynamic Formulation with Stress Analysis and a Non-linear FEA Code", *Int. J. for Numerical Methods in Engineering*, **Vol.** (51) (2001) p. 1033.

4. A. Hopgood, *Intelligent Systems for Engineers and Scientists*, CRC Press, New York, (2001).

5. Matlab 6.1 Documentation, *User's Guide Version 2.1*, (2000).

7. Agradecimientos

Este trabajo ha sido realizado en el contexto del proyecto DPI2000-0379, financiado por la CICYT, y del incentivo al mismo PGIDT01PXI16601PN aportado por la Secretaría General de I+D de la Xunta de Galicia.